CSCI 6510 - Distributed Systems and Algorithms Project 2

Team: Yichen Qiu, Guo Yu, Lili Li

Programming Language:
Python 2.7.6

Implementation Details:

In this project, we implemented a tweet-like service that utilized Paxos algorithm to reach consensus among multiple sites. Same as project1, this service implements basic functionalities like tweet, view timeline, block/unblock users. By using Paxos algorithm, this service maintain a uniform log that contains all events happened in every site.

Global Variables:

For config data of sites, we defined: *id_addr,* a dictionary data structure stores the site id and its corresponding ip address and port number, *tweets,* a list stores all tweets should be displayed in local site's timeline, *block_dict*, a dictionary stores a list of site id blocked by a site. *Logs*, a list of logs, *last_checked_logid* that is a max *log_id* already checked last time. We also defined *wating_queue* to events that needed to be reached consensus. For other related data of Paxos, we defined *largest_log_id, request_logid_max, request_logid_count, prepare_dict, prepare_ack_count and accept_ack_count*.

Threads:

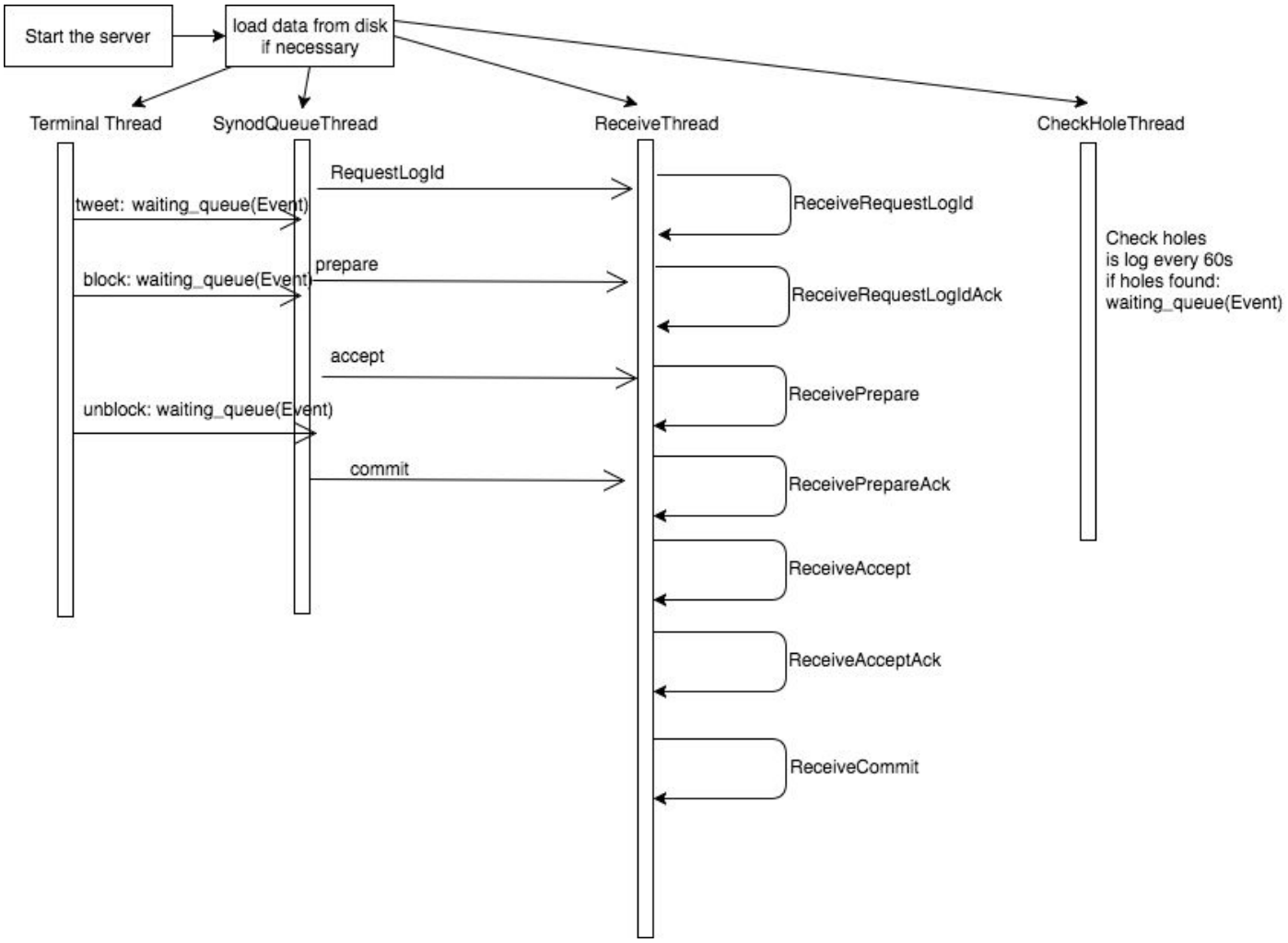
In server.py, we created four threads: *ReceiveThread*, *CheckHoleThread*, *SynodQueueThread* and *TerminalThread*

1. *ReceiveThread* is for receiving messages from other sites. In *ReceiveThread* we implemented different functions based on actions sent from other sites. After message was received by *ReceiveThread*, the action type will be extracted from message data and then the corresponding function will be called.
2. In *CheckHoleThread*, we check holes in logs of local site every 60 seconds. We check logs btw *last_checked_logid* and *request_logId_max* and put the missing event in *waiting_queue* to learn log event from other sites.
3. In *SynodQueueThread*, we implement the logic in Synod Algorithm. We broadcast event from *waiting_queue* one by one. For event popped out from *waiting_queue,* we sent request to all other sites to acquire the *logId* among other sites and set the *logId* of current event to the *request_logid_max*+1, and sent prepare request to sites. The *SynodQueueThread* will wait until majority sites response or timeout when each time it sends message to other sites. If the current event does not receive response from majority

sites, it will generate a new proposal id to get the priority. Same for accept and commit request, it will send out message to other sites and wait for the majority responses or time out to proceed to next stage or generate a new proposal id.

4. *TerminalThread* is for handling command read from terminal. It will execute specific requirements passed from terminal side: *tweet, view, block, unblock.*

The diagram below shows the basic architecture of the project:



Other Functionalities:

File recovery:

Whenever there's an update in *tweets*, *block_dict*, *logs*, *last_checked_logid, waiting_queue, prepare_dict* for current site, our program will dump these fields to local disk using python pickle. As the system executes, it will load all the global fields *tweets*, *logs,*

block_dict, prepare_dict, largest_log_id, last_checked_logid and other needed global variables from local disk if there exists a record for the current site. If not, these fields will be initialized as empty sets accordingly.

Deployment on AWS:

Three instances were setup in AWS. The instances was created in two different regions: US EAST (N. Ohio), US WEST (N. California). We assigned these three instances with port numbers: 8081, 8082 and 8083, 8084, 8085. These port numbers will be combined with the public IP of each instance in the system.